МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
Факультет прикладної математики та інформатики
Кафедра інформаційних систем

МАГІСТЕРСЬКА РОБОТА

«Застосування нейромереж для розпізнавання тексту та символів»

Виконала: студентка групи ПМІМ-22
спеціальності
122 Комп'ютерні науки
(шифр і назва спеціальності)

Кіндрат С.Я.
(підпис) (прізвище та ініціали)

Керівник Бернакевич І.Є.
(підпис) (прізвище та ініціали)

Рецензент Коковська Я.В.
(підпис) (прізвище та ініціали)

Львів 2022

ДЕКАН
Факультет прикладної математики та інформатики
ЛНУ ім. Івана Франка

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики _____
Кафедра Інформаційних систем _____
Спеціальність 122 Комп'ютерні науки _____
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____
" 5 " 09 2022 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Кіндрат Соломія Ярославівна
(прізвище, ім'я, по батькові)

1 .Тема роботи «Застосування нейромереж для розпізнавання тексту та символів» _____

керівник роботи Бернакевич Ірина Євстахіївна, доцент _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від " _____ " _____ 20 _ року № _____

2. Строк подання студентом роботи 12 грудня 2022р _____

3. Вихідні дані до роботи методи розпізнавання тексту, методи та алгоритми бібліотек ко комп'ютерного зору та бібліотек нейронних мереж, CNN, RNN, методи для спотворення зображень, мова програмування Python, бібліотеки OpenCV та PIL. _____

4. Зміст магістерської роботи (перелік питань, які потрібно розробити) _____

_____ Огляд, вивчення та аналіз існуючих систем для розпізнавання друкованого тексту; огляд методів розпізнавання тексту; ознайомлення з проблемами розпізнавання друкованого тексту; ознайомлення з методами та алгоритмами бібліотек комп'ютерного зору та бібліотек нейронних мереж; створення набору спотворених зображень в різний спосіб та використовуючи різні типи шрифтів для навчання моделі; розробка моделі оптичного розпізнавання тексту із зображення низької якості;

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1.	Формування завдання на магістерську роботу.	07.09.2022	
2.	Огляд технічної літератури за темою роботи.	31.09.2021	
3.	Аналіз актуальності теми.	15.10.2022	
4.	Дослідження теоретичної частини.	28.10.2022	
5.	Написання практичної частини.	16.11.2022	
6.	Написання теоретичної частини.	25.11.2022	
7.	Корегування роботи згідно з результатами перевірки роботи науковим керівником.	01.12.2022	
8.	Створення презентації.	07.12.2022	
9.	Подання роботи на кафедру для перевірки на плагіат.	12.12.2022	
10.	Захист магістерської роботи.	15-19.12.2022	

Студент Кіндрат Соломія Ярославівна _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Бернакевич І.Є.** _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Актуальність теми.

Актуальність теми дослідження зумовлено необхідністю автоматично розпізнавати друкований текст, оскільки існує ряд завдань введення великого об'єму текстової та графічної інформації в електронний формат. Разом з цим ці завдання є доволі складними, адже необхідно забезпечити високу надійність розпізнавання навіть при поганій якості друку чи зображення.

Нейронні мережі гарно зарекомендували себе у вирішенні завдань розпізнавання текстів та символів, де надійність розпізнавання багато в чому залежить від вибору структури мережі. Оскільки розвиток комп’ютерних технологій сягнув чималих висот, то стало можливим створення систем розпізнавання тексту, які б задовольняли основним вимогам автоматичного ведення друкованих документів.

Мета.

Метою даної роботи є розробка моделі для ефективного розпізнавання тексту, яка працюватиме із зображеннями низької якості.

Завдання.

Для досягнення поставленої вище мети були поставлені такі завдання:

- огляд, вивчення та аналіз існуючих систем для розпізнавання друкованого тексту;
- огляд методів розпізнавання тексту;
- ознайомлення з проблемами розпізнавання друкованого тексту;
- ознайомлення з методами та алгоритмами бібліотек комп’ютерного зору та бібліотек нейронних мереж;

- створення набору спотворених зображень в різний спосіб та використовуючи різні типи шрифтів для навчання моделі;
- розробка моделі оптичного розпізнавання тексту із зображення низької якості;
- тестування моделі розпізнавання тексту на валідаційному наборі зображень.

Об’єкт дослідження.

Об’єктом дослідження обрані сучасні системи OCR, які використовують нейронні мережі для розпізнавання тексту із зображення і перетворення його в електронну форму.

Предмет дослідження.

Предметом дослідження цієї роботи є методи розпізнавання тексту та символів, оптичне розпізнавання тексту, проблеми розпізнавання друкованого тексту, методи і алгоритми комп’ютерного зору та бібліотек нейронних мереж, мова програмування Python з її бібліотеками.

Ключові слова.

Розпізнавання тексту, OCR, нейронні мережі, CNN, RNN, методи і алгоритми комп’ютерного зору, OpenCV, PIL, Python, датасет.

ABSTRACT

Relevance of the topic.

The relevance of the research topic is determined by the need to automatically recognize printed text, as there are a number of tasks of entering a large volume of textual and graphic information into electronic format. At the same time, these tasks are quite difficult, because it is necessary to ensure high reliability of recognition even with poor print or image quality.

Neural networks have proven themselves well in solving text and character recognition tasks, where the reliability of recognition largely depends on the choice of network structure. Since the development of computer technologies has reached great heights, it has become possible to create text recognition systems that would satisfy the basic requirements of automatic maintenance of printed documents.

Purpose.

The purpose of this work is to develop a model for effective text recognition that will work with low-quality images.

Task.

To achieve the above purpose, the following tasks were set:

- review, study and analysis of existing systems for recognition of printed text;

- review of text recognition methods;

- familiarization with the problems of recognition of printed text;

- familiarization with methods and algorithms of computer vision libraries and neural network libraries;

- creating a set of distorted images in different ways and using different types of fonts for model training;

- development of a model of optical text recognition from a low-quality image;

- testing the text recognition model on a validation set of images.

The object of the study.

Modern OCR systems, which use neural networks to recognize text from an image and convert it into electronic form, are chosen as the object of research.

The subject of the study.

The subject of this work is text and character recognition methods, optical text recognition, printed text recognition problems, computer vision methods and algorithms and neural network libraries, the Python programming language and its libraries.

Key words.

Text recognition, OCR, neural networks, CNN, RNN, computer vision methods and algorithms, OpenCV, PIL, Python, dataset.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

OCR – Оптичне Розпізнавання Символів (Optical character recognition)

CNN – Згорткові Нейронні Мережі (Convolutional Neural Network)

RNN – Рекурентні Нейронні Мережі (Recurrent Neural Network)

MNIST – об'ємна база даних зразків рукописного написання цифр (Mixed National Institute of Standards and Technology)

ANN – Штучні Нейронні Мережі (Artificial Neural Networks)

OpenCV – Open Source Computer Vision Library

PIL – Python Imaging Library

ВСТУП

Стадія підготовки і обробки інформації, особливо при комп'ютеризації будь-якого підприємства – це важливий етап, якій вимагає багато зусиль та часу. Оскільки виникає завдання введення великого об'єму текстової і графічної інформації в електронний формат. Сканер, факс-модем, цифрова фотокамера – це основні пристрої для введення графічної інформації. Завдання розпізнавання текстової інформації є одним з найважливіших завдань, що мають на меті автоматизацію документообігу. Разом з тим ця проблема є однією з найбільш складних в області автоматичного аналізу зображень. Що стосується систем розпізнавання важливих друкованих документів, то тут складність полягає в тому, що необхідно забезпечити високу надійність розпізнавання (більше 98-99%) навіть при поганій якості зображення.

Для вирішення завдань розпізнавання текстів і символів останнім часом широко використовуються нейронні мережі, де надійність розпізнавання багато в чому залежить від вибору структури і параметрів мережі. Завдяки використанню сучасних досягнень комп'ютерних технологій, були розвинені нові методи обробки зображень і розпізнавання образів, завдяки чому стало можливим створення систем розпізнавання друкованого тексту, які задовольняли б основним вимогам систем автоматизації документообігу.

Сучасні OCR системи використовують нейронні мережі. OCR, або оптичне розпізнавання символів, — це технологія для розпізнавання тексту із зображень і перетворення його в електронну форму. Ці зображення можуть бути друкованим чи рукописним текстом, як-от, наприклад, документи.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД OCR СИСТЕМ

1.1. Огляд стану проблеми та постановка задачі

Проблема ефективного розпізнавання тексту посідає важливе місце в сферах інформатизації різних процесів людської діяльності. Текстове представлення інформації, порівняно із графічним, дозволяє істотно скоротити витрати на зберігання та передачу інформації, а також дозволяє реалізувати всі методи використання та аналізу електронних документів. Тому найбільший інтерес з практичної точки зору являє саме перетворення інформації з паперових носіїв в текстовий електронний документ.

У сучасному світі люди частіше використовують камеру мобільного телефону, замість сканера, що ще більше ускладнює проблему ефективного розпізнавання тексту. Оскільки зображення, що зробленою фото-камерою, може містити чимало спотворень, наприклад, таких як цифровий шум, зернистість, розмиття та чимало інших проблем. Для розпізнавання повного тексту, необхідно розпізнати на зображенні окремі символи. Існує чимало програмних продуктів, які практично автоматизували процеси розпізнавання текстів. Проте забезпечити задовільний результат у випадку спотворень зображень текстів різного типу вдається не завжди. Проте варто зазначити, що здатність людини читати друкований текст низької якості перевершує можливості комп’ютерних систем.

Проблеми ідентифікації букв та символів можуть виникати у зв’язку з візуальною подібністю. Наприклад, погано розпізнаються букви чи цифри такі, як «t», «f», «l», «i», «G», «6», «5», «U».

Метою даної роботи є розробка моделі для ефективного розпізнавання тексту, яка працюватиме із зображеннями низької якості.

Для досягнення даної мети були поставлені такі завдання:

- огляд, вивчення та аналіз існуючих систем для розпізнавання друкованого тексту;
- огляд методів розпізнавання тексту;
- ознайомлення з проблемами розпізнавання друкованого тексту;
- ознайомлення з методами та алгоритмами бібліотек комп’ютерного зору та бібліотек нейронних мереж;
- створення набору спотворених зображень в різний спосіб та використовуючи різні типи шрифтів для навчання моделі;
- розробка моделі оптичного розпізнавання тексту із зображення низької якості;
- тестування моделі розпізнавання тексту на валідаційному наборі зображень.

1.2. Технологія OCR

Оптичне розпізнавання тексту (англ. optical character recognition, OCR) — це механічне або електронне перетворення зображень рукописного, машинописного або друкованого тексту в послідовність кодів, що використовуються для представлення в текстовому форматі.[1] Розпізнавання широко застосовують для атоматичного перетворення в електронний формат документів, книг або для публікації тексту на веб-сторінка. OCR дозволяє редагувати, форматувати та аналізувати текст, застосовувати до нього електронний переклад, здійснювати пошук по ньому, а також зберігати його в різних форматах.

Оптичне розпізнавання тексту є досліджуваною проблемою в галузях розпізнавання образів, штучного інтелекту і комп'ютерного зору.[1]

Головним завданням роботи OCR є вирішення проблеми, яка полягає у тому, що існує багато способів написання одного і того ж символу та різних шрифтів. Окрім того, перш ніж обирати алгоритм OCR, зображення повинно мати попередню обробку, щоб воно було готовим до «читання». У багатьох

випадка система OCR виконує цей етап самостійно, щоб забезпечити високу надійність розпізнавання.

Як правило, процес відбувається приблизно так:

1. Документи, скануються або фотографуються.
2. Застосовується технологія оптичного розпізнавання символів.
3. Інформація, зберігається у вигляді тексту з метаданими для пошуку.
4. Отримані дані використовуються для визначених цілей.

Останні дослідження технології оптичного розпізнавання тексту базуються на застосуванні принципів роботи зорової системи людини (IPA) [2], таких як:

– принцип цілісності (integrity), згідно з яким об’єкт представлено як сукупність частин і просторових взаємозв’язків між ними;

– принцип цілеспрямованості (purposefulness): будь-яка інтерпретація даних переслідує певну мету. Згідно з цим принципом, розпізнавання являє собою процес висунення гіпотез про цілий об’єкт і цілеспрямованої їх перевірки;

– принцип адаптивності (adaptability) передбачає здатність системи до самонавчання шляхом збереження інформації в процесі її обробки.

Переваги системи розпізнавання, що працює відповідно до принципів IPA, очевидні: саме вони здатні забезпечити максимально гнучку і осмислену поведінку системи.

Механізми OCR розроблені для багатьох галузевих застосувань, включаючи оцифровування книг.

Наведемо типові випадки використання:

- Введення даних для документів, наприклад, чеків, квитанцій, паспортів, рахунків, банківських виписок;
- Автоматичне розпізнавання номерних знаків;
- Розпізнавання паспорта та отримання інформації;
- Автоматичне вилучення ключової інформації страхового документа;

- Вилучення інформації про візитну картку до списку контактів;
- Створення цифрової версії друкованого документа, наприклад, книги;
- Перетворення відсканованих зображень друкованих документів у електронний формат;

Переваги OCR-систем:

1. *Доступність*. Коли сканований документ OCR потрапляє до загальної бази даних, він стає доступним для всіх користувачів, у яких є доступ до цієї бази даних.
2. *Доступність для пошуку*. OCR-системи дозволяють зберегти відсканований файл у форматі .pdf, .doc, .rtf, .txt (найпростіший) тощо після того, як файл буде відскановано у читабельний текст. В результаті ці файли дозволяють легко проводити пошук по них.
3. *Можливість редагування*. У випадку необхідності, наприклад, переглянути або змінити текст, то після оцифровування цього документа за допомогою OCR, з'являється можливість редагувати його за допомогою текстового редактора, а не вводити весь текст документу заново.
4. *Компактне зберігання.* Перетворення тексту в електронний формат зменшує фізичний простір, необхідний для зберігання архівів документів із однієї або кількох кімнат до декількох гігабайтів на сервері. Це забезпечує більшу продуктивність. Окрім цього, сьогодні паперові архіви можна переробити і не накопичувати тони паперу.
5. *Резервне копіювання*. Замість того, щоб зберігати паперові дублікати та копії, цифрові резервні копії можна зробити доволі дешево та швидко у необмеженій кількості.

1.3. Методи розпізнавання тексту

Методи розпізнавання тексту та символів різних зображень забезпечує вирішення ряду наукових та прикладних задач при ідентифікації об'єктів. Сучасні методи розпізнавання символів використовуються для вирішення як типових, так

і різних спеціалізованих задач. Прикладом типової задачі може бути розпізнавання друкованого тексту, а прикладом спеціалізованої задачі може бути розпізнавання спеціальної символьної інформації. Існує велика кількість програм, призначених для розпізнавання тексту. Такі як FineReader, ScanSoft OmniPage та інші. Кожна із запропонованих програм надає свою реалізацію для вирішення задачі обробки та розпізнавання зображень.

Основні методи для розпізнавання тексту [3]:

1. *Структурні методи* розпізнавання зберігають інформацію не про поточкове написання символу, а про його топологію. Еталон містить інформацію про взаємне розташування окремих складових частин символу. Перевага методу – стійкість до зсуву і повороту символу на невеликий кут, до різних стильових варіацій шрифтів. Однак, при повороті на кут, більший десяти градусів, даний метод не може бути використаний для розпізнавання символів. При застосування цього методу неважливими стають такі ознаки як розмір букви, що розпізнається і навіть шрифт, яким вона надрукована. Проте, основною проблемою цього методу є ідентифікація знаків, які містять певні дефекти (наприклад, розрив ліній або з'єднання сусідніх ліній).
2. *Шаблонні методи* перетворюють зображення окремого символу в растрове, порівнюють його зі всіма шаблонами, наявними в базі і вибирають шаблон з найменшою кількістю крапок, відмінних від вхідного зображення. Шаблонні методи досить стійкі до дефектів зображення і мають високу швидкість обробки вхідних даних, але надійно розпізнають тільки ті шрифти, шаблони яких їм «відомі». І якщо розпізнаний шрифт хоч трохи відрізняється від еталонного, шаблонні методи можуть робити помилки навіть при обробці дуже якісних зображень.
3. *Ознакові методи* базуються на тому, що зображенню ставиться у відповідність N-мірний вектор ознак. Розпізнавання полягає в порівнянні

вектора ознак з набором еталонних векторів тієї ж розмірності. Переваги методу – простота реалізації, хороша узагальнююча здатність, висока швидкість розпізнавання. Недоліком методу є висока чутливість до дефектів зображення. Крім того, ознакові методи мають інший недолік — на етапі виділяння ознак відбувається незворотня втрата частини інформації про символ. Виділяння ознак проходить незалежно, тому інформація про взаємне розташування елементів символів втрачається.

1.4. Методи машинного навчання

Задачу оптичного розпізнавання тексту можна вирішувати за допомогою методів машинного навчання, а саме навчання з учителем.

Навчання з учителем передбачає наявність певної кількості об'єктів X, які представлені рядом ознак, і безлічі відповідей або результатів Y, які також називаються мітками об'єктів. Між цими множинами існує якась залежність. Залежність відповідей від об'єктів відома тільки на об'єктах з навчальної вибірки.

Завдання навчання з учителем полягає в тому, щоб відновити залежність, тобто навчитися передбачати відповіді $y \in Y$ по нових об'єктах $x \in X$.

Об'єкти з безлічі X описуються рядом ознак. Кожна ознака об'єкта також має деякі властивості: $x \in X = Q_1 \dots Qn$:

- якщо Qj = R, то ознака кількісна;
- якщо Qj звичайна, то ознака є номінальною (при цьому, якщо |Qj|=2, то така ознака називається бінарною);
- якщо Qj звичайна і впорядкована, то така ознака називається порядкованою.

Існує кілька способів створення простору ознак опису об'єктів:

- відбір значимих ознак;
- синтез нових ознак з уже існуючих.

Якщо перший спосіб цілком можна виконати вручну, за допомогою фахівців в конкретній предметній області, то другий краще довірити машині, щоб вона сама вирішила, які ознаки необхідно синтезувати для того, щоб отримати більш точний результат.

Інформацію про об'єкти зручно записувати у вигляді матриці, кожен рядок якої - це об'єкт, а кожен стовпець - це ознака або атрибут. Очікувані відповіді також записують у вигляді матриці. Виходячи з безлічі Y, можна виділити кілька типів завдання навчання:

- якщо Y звичайне, то це завдання класифікації. Часто використовується, наприклад, в задачах розпізнавання. У підсумку, завдання машини зводиться до того, щоб визначити, до якого класу належить об'єкт x;
- якщо Y = R, то це завдання відновлення регресії. Таке завдання зводиться до визначення функції f з певного класу, яка апроксимує невідому залежність.

Методи машинного навчання з учителем:

- логістична регресія;
- метод найближчих сусідів;
- метод опорних векторів;
- нейронні мережі;
- інші.

1.5. Нейронні мережі

Сучасні OCR системи побудована із використанням нейронних мереж, що допомагає вирішувати складні задачі машинного навчання.

Нейронна мережа складається з елементарних одиниць - нейронів. Кожен нейрон має кілька входів і один вихід. На рисунку 1.1 представлена модель нейрона, яка була запропонована Маккалоком-Піттсом в 1943 році.

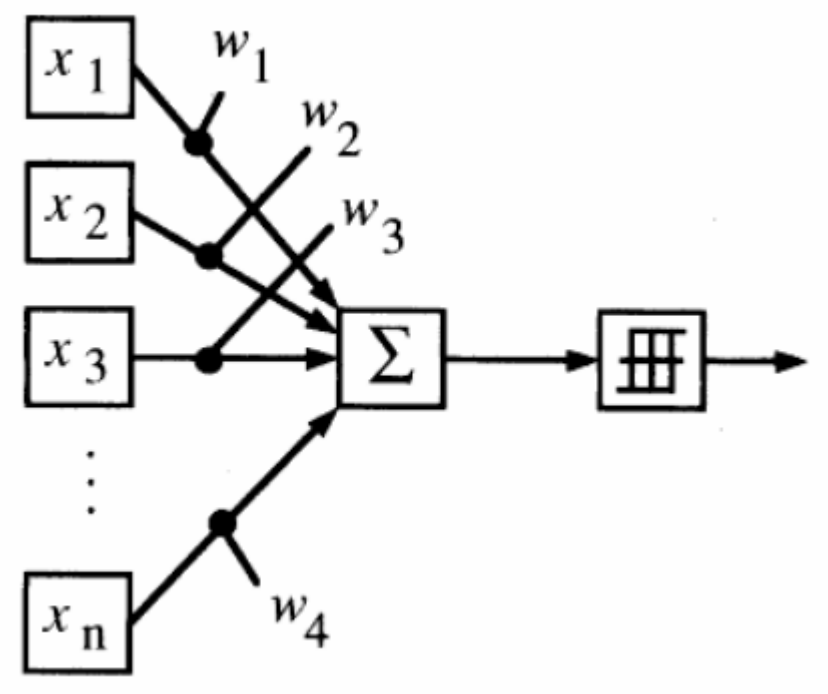

Рис. 1.1. – Модель штучного нейрона

На рисунку 1.2 представлена модель біологічного нейрона.

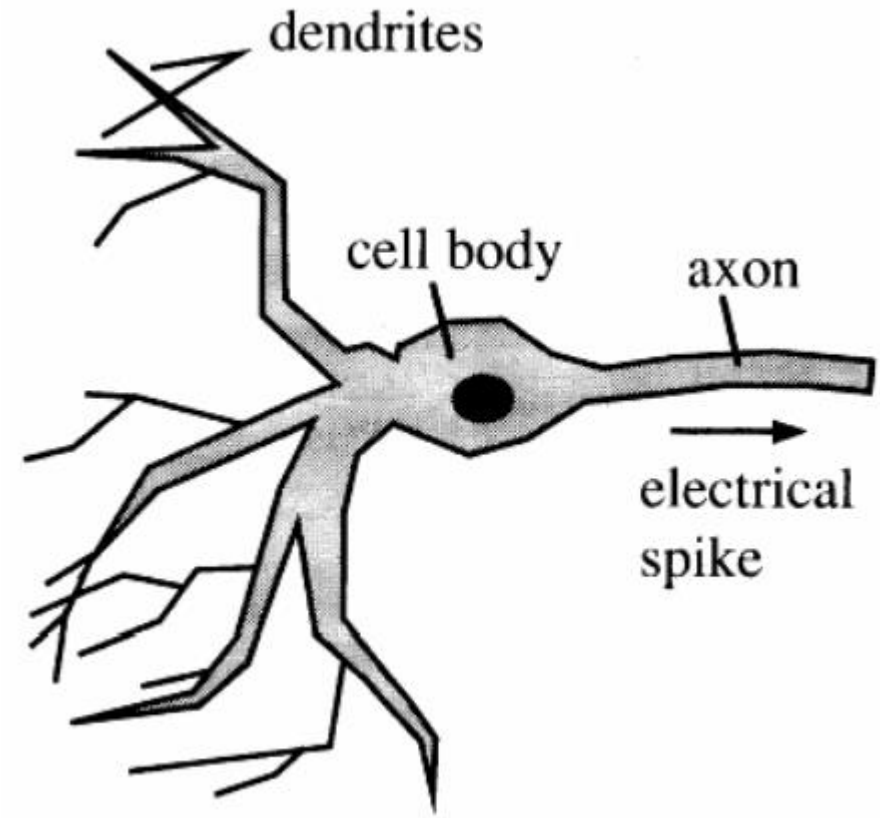

Рис. 1.2. – Модель біологічного нейрона

Основними параметрами нейрона є ваги і функція активації. Зазвичай ваги представляють у вигляді деякого вектора $\begin{bmatrix} w_1 \\ \cdots \\ w_n \end{bmatrix}$.

Сам по собі принцип роботи спрощеної моделі нейрона простий:

- на вхід подається набір параметрів $[x_1, \quad \cdots, \quad x_n]$;
- параметри підсумовуються з урахуванням представлених ваг: $\sum_i^n w_i x_i$;
- отримане значення передається в якості аргументу функції активації, яка повертає підсумкове значення $f(\sum_i^n w_i x_i)$.

Пізніше в 1949 році Дональдом Хеббом був запропонований перший алгоритм навчання і ним же сформульовані правила:

- якщо вихідний сигнал (результат) є невірним і дорівнює нулю, то необхідно збільшити значення синоптичних зв'язків;
- якщо вихідний сигнал (результат) є невірним і дорівнює одиниці, то необхідно зменшити значення синоптичних зв'язків.

Незважаючи на те, що технології пішли далеко вперед, дані правила все ще актуальні і лежать в основі багатьох проектованих нейронних мережах.

Важливою подією в нейромережевих технологіях стала поява персептрона Розенблатта в 1958 році, який складався з елементів трьох типів:

- S-елементи - це шар рецепторів;
- A-елементи - з'єднані з S-елементами. Є суматорами з деякою пороговою функцією. Фактично кожен A-елемент являє собою нейрон. При перетині порогового значення, сигнал з A-елемента передається на вхід R-елемента з урахуванням вагового коефіцієнта;
- R-елементи - це суматори, які підсумовують значення вхідних сигналів з урахуванням ваг і також передають знайдене значення порогової функції. Від результату функції залежить вихідне значення.

Персептрон Розенблатта вже був здатний вирішувати такі завдання, як класифікація, розпізнавання і узагальнення. Також у Розенблатта була описана модель багатошарового персептрона з декількома шарами А-елементів.

Для навчання як одношарового, так і багатошарового перспетрона, Розенблатт користувався правилами Хебба. Навчання багатошарового персептрона зажадало деякої модифікації функції активації, а саме використання диференціювання, для того, щоб було можливо використовувати метод градієнтного спуску. Найбільш частим прикладом такої функції є сигмоїда. Важливою особливістю в навчанні персептрона було те, що навчатися могли не всі верстви, а лише деякі з них, в той час як ваги решти шарів були жорстко фіксовані.

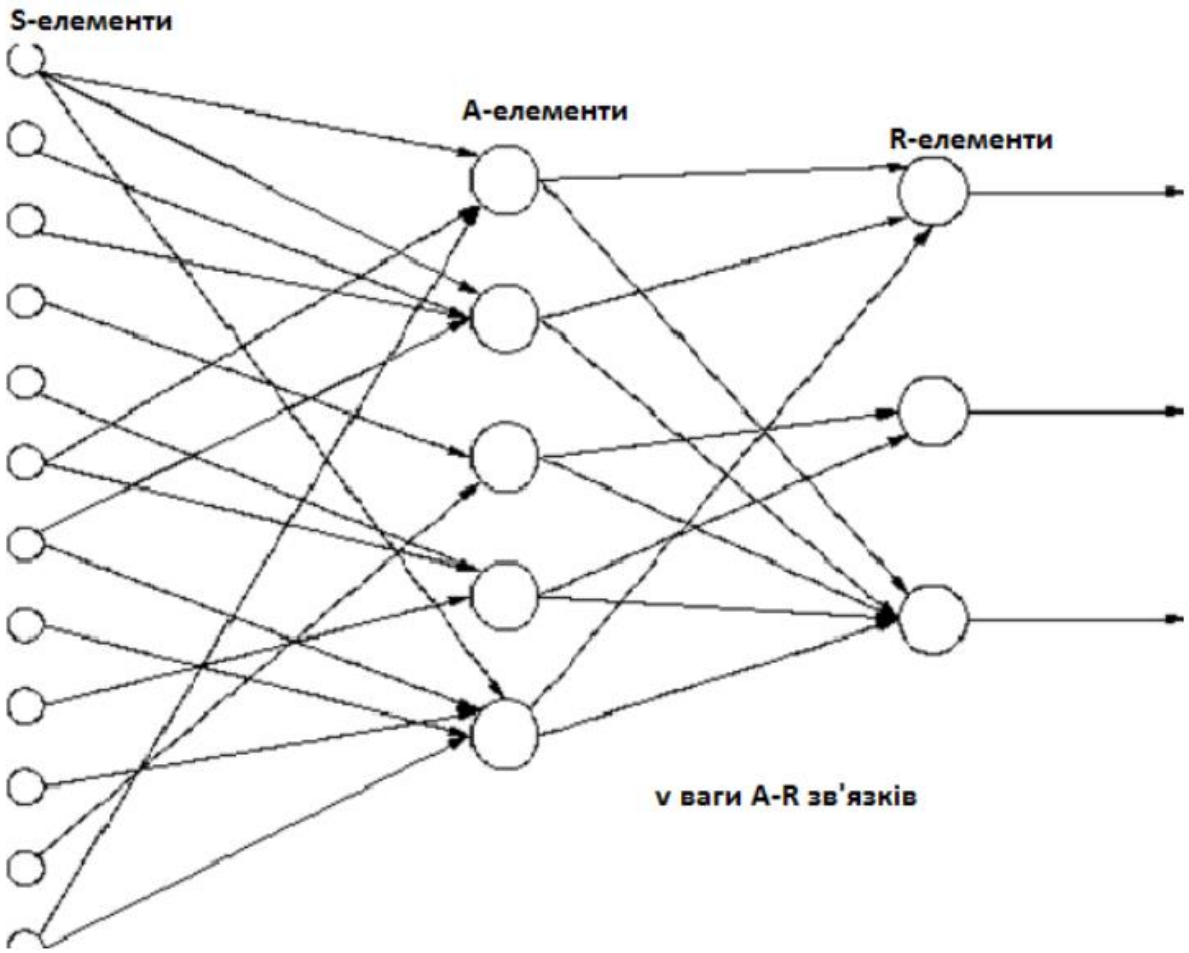

Рис. 1.3. – Одношаровий персептрон

У 1989 році групою вчених, серед яких були Девід І. Румельхарт, Дж. Е. Хінтон і Рональд Дж. Вільямс, був розроблений метод навчання, який отримав назву «метод зворотного поширення помилки». До розробки даного методу було досить-таки складно налаштовувати ваги нейронних мереж з одними і більш прихованими шарами.

Згорткові нейронні мережі (CNN) - це клас штучних нейронних мереж, які є більш вдосконаленими моделями порівняно з багатошаровим персептроном. У CNN об'єднуються згорткові шари і так звані шари пулінгу, що використовуються для зменшення розмірності зображення, з повнозв'язними шарами, використовуваними в багатошаровому персептроні. Шари згортки і пулінгу необхідні для отримання ознак з вхідних даних, а повнозв'язні шари використовуються для перетворення витягнутих і оброблених ознак у вихідні сигнали мережі. Згортковий шар виконує математичну операцію, яку називають згорткою. В наборі даних MNIST всі цифрові зображення зберігаються в двовимірному форматі. Двовимірна матриця меншого розміру, в термінах згорткових нейронних мережах називається ядром, або фільтром, що використовується для виконання операції згортки з кожним нейроном згорткового шару. По суті, згортка - це скалярний добуток між ядром і вхідним тензором. Доповнюючи нулі в масивах зображень зазвичай необхідні для того, щоб зберегти пропорції в розмірах фільтра і тензора. Шари пулінгу дозволяють зменшити розмірність зображення та використовувати меншу кількість параметрів. У CNN використовується алгоритм зворотного поширення помилки для вивчення просторової ієрархії ознак й коригування ваг нейронів, так як це і відбувається в архітектурі багатошарового персептрону. Як правило, для навчання моделі CNN потрібне використання графічних процесорів через велику кількість параметрів, що налаштовуються. Згорткові нейронні мережі завжди були основою глибинного навчання. На рисунку 1.4 наведено простий приклад архітектури згорткових нейронних мереж, призначеної для завдання класифікації

цифр на зображеннях. Для цього набору даних є десять класів, таким чином, загальне число вихідних сигналів мережі дорівнює десяти.

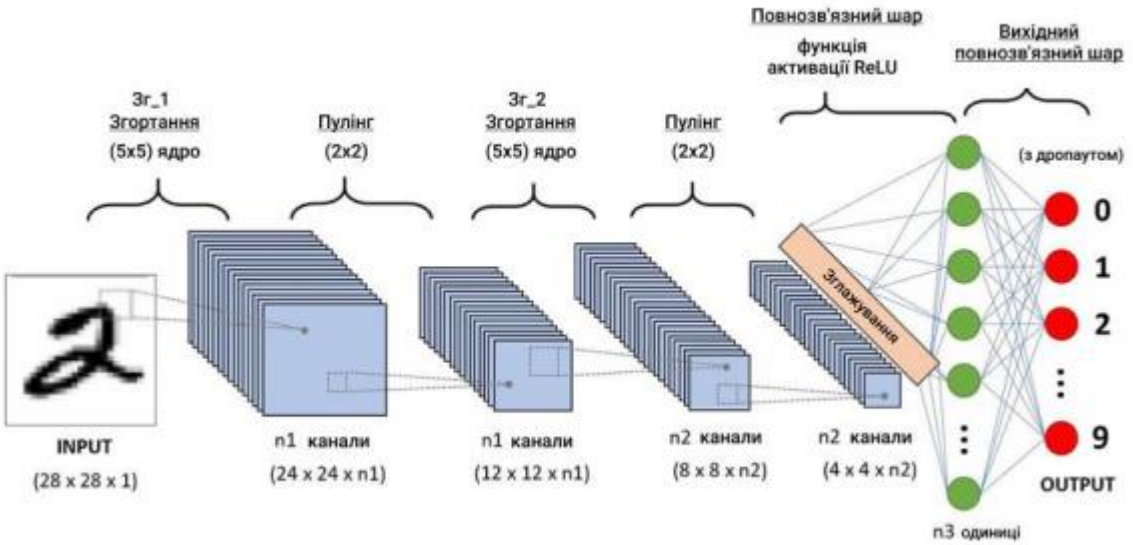

Рис. 1.4. – Приклад архітектури згорткової нейронної мережі

Рекурентна нейронна мережа (RNN) - це клас вдосконаленої штучної нейронної мережі (ANN), що містять зворотні зв'язки та дозволяють зберігати інформацію (див. рисунок 1.5).

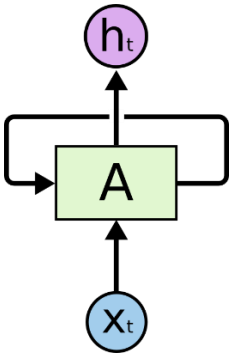

Рис. 1.5 - Рекурентні нейронні мережі містять зворотні зв'язки.

На схемі вище зображений фрагмент нейронної мережі A, який приймає вхідне значення x_t і повертає значення h_t. Наявність зворотного зв'язку дозволяє передавати інформацію від одного кроку мережі до іншого.

Зворотність зв'язку надають рекурентним нейронних мереж (RNN) загадковості. Проте, якщо все зважити, вони не дуже сильно відрізняються від звичайних нейронних мереж. Рекурентну мережу можна розглядати, як кілька копій однієї і тієї ж мережі, кожна з яких передає інформацію наступній копії.

Ось, що станеться, якщо ми розгорнемо зворотний зв'язок, представлено нижче на рисунку 1.5:

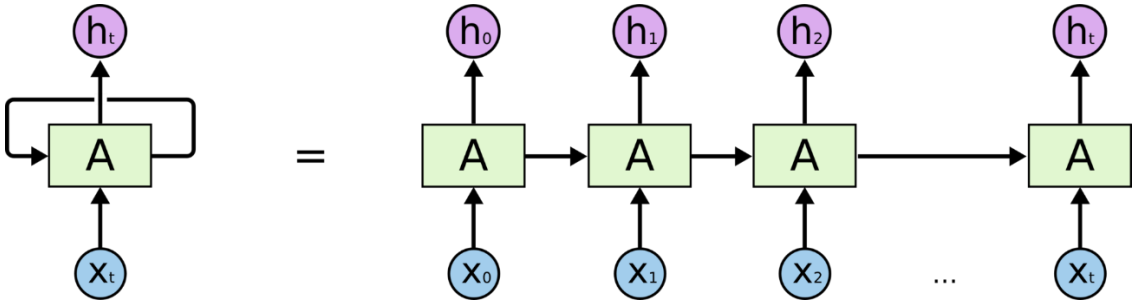

Рис. 1.6 - Рекурентна нейронна мережа в розгортці

Рекурентні нейронні мережі (RNN) нагадують ланцюжок, що свідчить про те, що вони тісно пов'язані з послідовностями і списками. За останні кілька років RNN з неймовірним успіхом застосували до цілого ряду завдань: розпізнавання мови, мовне моделювання, переклад, розпізнавання зображень тощо.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ

2.1. Обробка зображень

2.1.1. OpenCV

OpenCV (Open Source Computer Vision Library) — це величезна бібліотека з відкритим вихідним кодом для комп'ютерного зору, машинного навчання та обробки зображень, і тепер вона відіграє важливу роль, що дуже важливо в сучасних системах. Використовуючи його, можна обробляти зображення та відео, щоб ідентифікувати об'єкти, обличчя чи навіть почерк людини. Після інтеграції з різними бібліотеками, такими як NumPy, python здатний обробляти структуру масиву OpenCV для аналізу. Для визначення візерунка зображення та його різних ознак ми використовуємо векторний простір і виконуємо математичні операції над цими ознаками.

Перша версія OpenCV була 1.0. OpenCV випускається під ліцензією BSD, тому він безкоштовний як для академічного, так і для комерційного використання. Він має інтерфейси C++, C, Python і Java і підтримує Windows, Linux, Mac OS, iOS і Android. Під час розробки OpenCV основну увагу приділяли додаткам реального часу для підвищення ефективності обчислень. Усе написано на оптимізованому C/C++, щоб скористатися перевагами багатоядерної обробки.

2.1.2. PIL

PIL (Python Imaging Library) — це безкоштовна додаткова бібліотека з відкритим вихідним кодом для мови програмування Python, яка надає можливості для відкриття, керування та збереження багатьох різних форматів файлів зображень. Він доступний для Windows, Mac OS X і Linux. Остання версія PIL — 1.1.7, випущена у вересні 2009 року і підтримує Python 1.5.2–2.7.

Розробка оригінального проекту, відомого як PIL, була припинена в 2011 році. Згодом наступний проект під назвою Pillow роздвоїв репозиторій PIL і

додав підтримку Python 3.x. Цей форк був прийнятий як заміна оригінального PIL у дистрибутивах Linux, включаючи Debian та Ubuntu (з 13.04).

Цей приклад завантажує зображення з файлової системи, розмиває його та показує на екрані як оригінал, так і розмите зображення:

```
from PIL import Image, ImageFilter  # Import classes from the library.

original_image = Image.open("file.png") # Load an image from the file
system.
blurred_image = original_image.filter(ImageFilter.BLUR) # Blur the image.

# Display both images.
original_image.show()
blurred_image.show()
```

2.1.3. Розмиття зображення

Зображення виглядає чіткішим або більш детальним, якщо ми вміємо правильно сприймати на ньому всі об’єкти та їх форми. Наприклад, зображення з обличчям виглядає чітким, коли ми можемо дуже чітко розпізнати очі, вуха, ніс, губи, чоло тощо. Така форма предмета обумовлена його краями. Завдання розмиття зображення полягає у тому, щоб зробити перехід від одного кольору до іншого дуже плавним.

Типи розмиття:

- *Розмиття за Гаусом (Gaussian blur)* - це одни з найбільш використовуваних фільтрів при обробці зображень. Він використовує гаусову криву розподілу, що визначається такою функцією:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}.$$

- *Медіанний фільтр (Median filter)* - це дуже простий фільтр, який повертає середнє значення з пікселя та його сусідів.

- *Розмиття коробки (Box blur)* — це лінійний фільтр просторової області, в якому кожен піксель в отриманому зображенні має значення, рівне середньому значенню сусідніх пікселів у вхідному зображенні. Розмиття поля 3x3 можна записати як матрицю:

$$\frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}.$$

- *Макс/Мін фільтр (Max/min filter)* розмиває зображення, замінюючи кожен піксель різницею між найвищим пікселем і найнижчим пікселем (відносно інтенсивності) у межах заданого розміру вікна. Наприклад, враховуючи краї 3x3 пікселів у відтінках сірого.

$$\begin{pmatrix}22 & 77 & 48\\ 150 & 77 & 158\\ 10 & 77 & 219\end{pmatrix}$$

Центральний піксель буде змінено з 77 на 209, оскільки це різниця між найяскравішим пікселем 219 і найтемнішим пікселем 10 у поточному вікні.

2.1.4. Цифровий шум зображення

Шум зазвичай визначається як випадкова зміна інформації про яскравість або колір і часто виникає внаслідок технічних обмежень датчика збирання зображень або неналежних умов навколишнього середовища. Ці труднощі часто неминучі в реальних сценаріях, що робить шум зображення поширеною проблемою, яку необхідно вирішувати за допомогою відповідних підходів до шумозаглушення.

Позбавлення від шуму зображення є складним завданням, оскільки шум пов’язаний з високочастотним вмістом зображення, тобто деталями. В результаті мета полягає в тому, щоб досягти балансу між придушенням шуму якомога сильніше, не втрачаючи занадто багато інформації. Підходи, засновані на фільтрах, для зменшення шуму зображення, такі як зворотний, медіанний і фільтр Вінера, найчастіше використовуються.

Наявність шуму на зображенні може бути адитивним або мультиплікативним. У моделі адитивного шуму до вихідного сигналу додається сигнал адитивного шуму, щоб створити пошкоджений шумовий сигнал, який відповідає наступному правилу:

$$w(x, y) = s(x,y) + n(x,y)$$

Тут s(x,y) представляє вихідну інтенсивність зображення, а n(x,y) представляє шум, який подається для створення пошкодженого сигналу w(x,y) у позиції (x,y) пікселя. Аналогічно, модель мультиплікативного шуму помножує вихідний сигнал на сигнал шуму.

Характер шуму, а також його ймовірні властивості відрізняють його. Існує широкий спектр типів шуму. Наприклад:

- *Гаусів шум (Gaussian noise)* - це статистичний шум з функцією щільності ймовірності, що є рівною нормальному розподілу. Гаусів шум має рівномірний розподіл по всьому сигналу. Зображення із шумом містить пікселі, які складаються із суми вихідних значень пікселів плюс випадкове значення гаусового шуму. Функція розподілу ймовірності для гаусового розподілу має форму дзвону. Адитивний білий гаусів шум є найпоширенішим застосуванням гаусового шуму в програмах. На рисунку 2.1. показано функцію розподілу Гауса (функція розподілу ймовірностей) та піксельне представлення гаусового шуму.

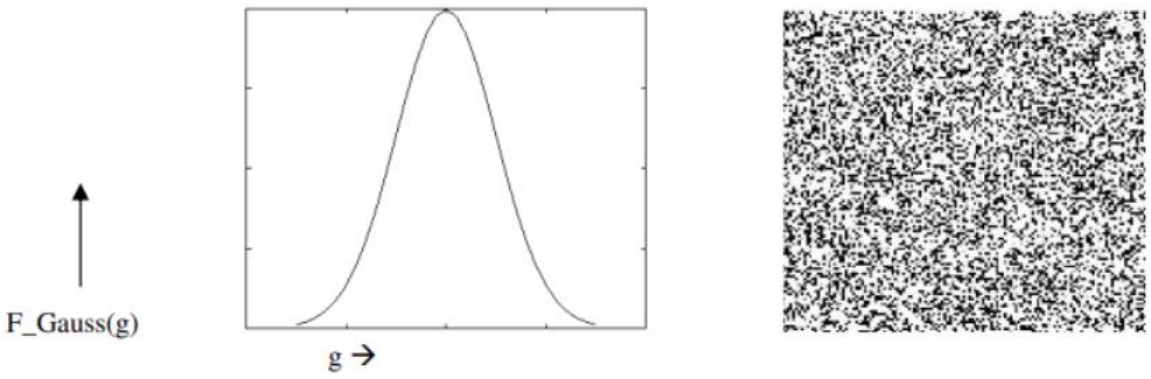

Рис. 2.1. - Функція розподілу ймовірностей та Гаусів шум

- *Шум солі та перцю (Salt and Pepper Noise)* – це тип шуму, який зазвичай можна побачити на фотографіях. Він проявляється у вигляді білих і чорних пікселів, які з’являються через випадкові інтервали. Помилки під час передачі даних викликають цей вид шуму. Значення a і b в шумі солі перцю різні. Кожна з них має в середньому ймовірність менше 0,1. Пошкоджені пікселі по черзі встановлюються як на мінімальне, так і на найвище значення, що надає зображенню «сіль і перець». Розподіл і піксельне представлення цього шуму показано на рисунку 2.2.

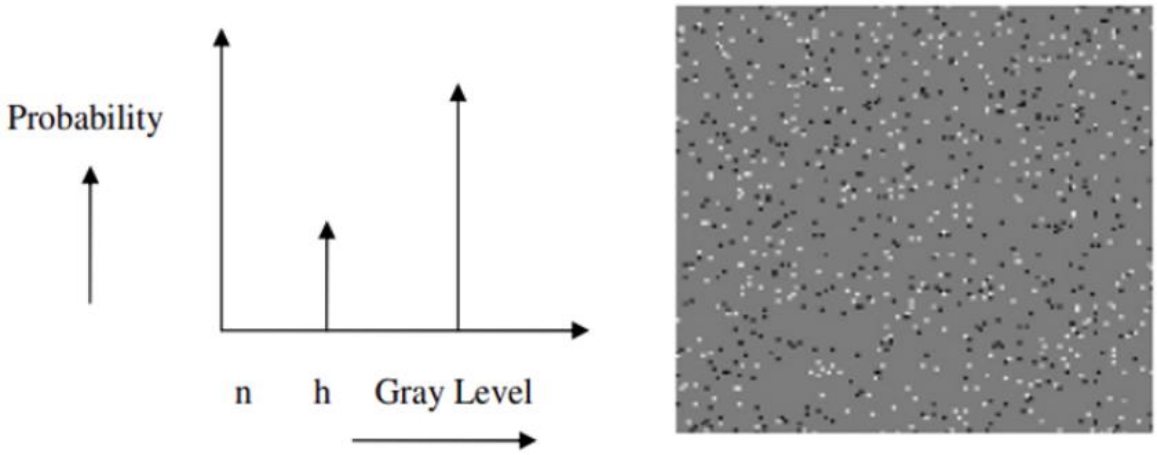

Рис. 2.2. - Функція розподілу ймовірностей та шум солі та перцю

- *Скляний шум (Speckle noise)* – це мультиплікативний шум. При діагностичних дослідженнях це знижує якість зображення, надаючи зображенням вигляд хвиль зі зворотнім розсіюванням, викликаний багатьма мікроскопічними розсіяними відображеннями, що протікають через внутрішні органи. Це ускладнює для спостерігача розрізнення дрібних деталей на зображеннях. Розподіл і піксельне представлення цього шуму показано нижче.

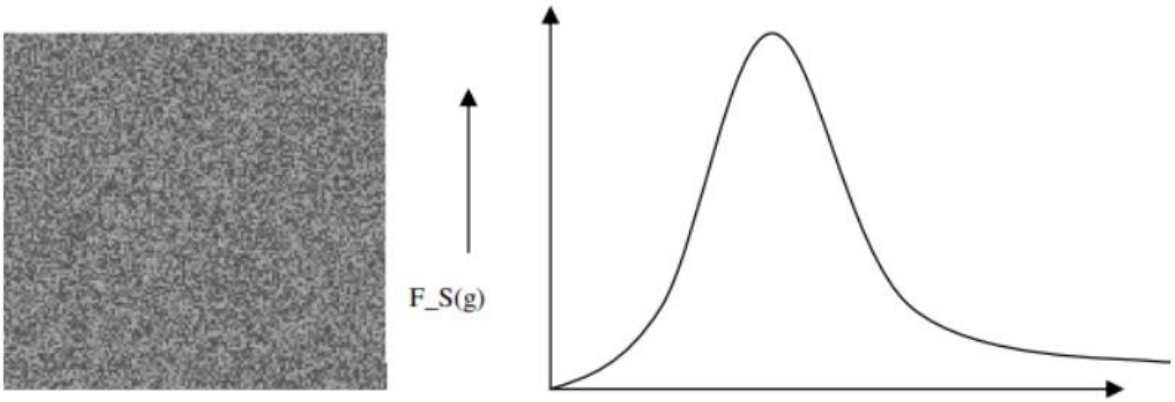

Рис. 2.3. - Функція розподілу ймовірностей та скляний шум

- *Пуассонівський шум (Poisson noise)* створюється в результаті нелінійних реакцій детекторів зображень і записувачів. Цей тип шуму визначається даними зображення. Оскільки процедури виявлення та запису включають довільну електронну емісію з розподілом Пуассона та середнім значенням відповіді, цей вираз використовується. Оскільки середнє значення і дисперсія розподілу Пуассона однакові, вважається, що залежний від зображення термін має стандартне відхилення, якщо припускається, що шум має дисперсію одиницю.

2.2. TensorFlow

TensorFlow — це безкоштовна бібліотека програмного забезпечення з відкритим вихідним кодом для машинного навчання та штучного інтелекту. Його можна використовувати в ряді завдань, але він зосереджений на навчанні та припущені глибоких нейронних мереж.

TensorFlow був розроблений командою Google Brain для внутрішнього використання Google у дослідженнях. Початкова версія була випущена під ліцензією Apache License 2.0 у 2015 році. Google випустила оновлену версію TensorFlow під назвою TensorFlow 2.0 у вересні 2019 року.

TensorFlow можна використовувати в широкому спектрі мов програмування, особливо в Python, а також у Javascript, C++ і Java. Ця гнучкість підходить для широкого спектру застосувань у багатьох різних секторах.

2.2.1. Keras

Keras — це ефективний високорівневий інтерфейс програмування прикладних програм (API), написаний на мові Python. Ця бібліотека нейронної мережі з відкритим кодом розроблена для швидкого експериментування з глибокими нейронними мережами, і вона може працювати поверх CNTK, TensorFlow і Theano.

Keras зосереджується на тому, щоб бути модульним та зручним. Він не обробляє низькорівневі обчислення, а замість цього він передає їх до іншої бібліотеки під назвою Backend.

Keras був прийнятий і інтегрований в TensorFlow в середині 2017 року. Користувачі можуть отримати доступ до нього через модуль tf.keras. Проте бібліотека Keras все ще може працювати окремо та незалежно.

Наступні вісім кроків – це кроки для створення моделі глибокого навчання в Keras:

1. Завантаження даних
2. Попередня обробка завантажених даних
3. Визначення моделі
4. Компіляція моделі
5. Навчання вказаної моделі
6. Оцінка
7. Прогнози
8. Збереження моделі

Шари Keras є основним будівельним блоком моделей Keras. Кожен рівень отримує вхідну інформацію, виконує деякі обчислення і, нарешті, виводить

перетворену інформацію. Вихід одного шару буде перетікати в наступний шар як його вхід.

До використаних в цій роботі шарів API належать:

- tf.keras.layers.Input - використовується для створення екземпляра тензора Keras;
- tf.keras.layers.Conv2D - шар двовимірної згортки;
- tf.keras.layers.MaxPooling2D - максимальна операція об’єднання для 2D просторових даних;
- tf.keras.layers.Reshape - шар, який змінює форму входу у задану форму;
- tf.keras.layers.Dense - повнозв’язний шар нейронної мережі;
- tf.keras.layers.Dropout - шар Dropout випадково встановлює значення 0 частині вхідних даних з частотою кроку протягом часу навчання, що допомагає запобігти перенавчанню. Входи, не встановлені на 0, збільшуються на 1/(1 - швидкість), так що сума по всіх входів не змінюється;
- tf.keras.layers.Bidirectional - двонаправлена оболонка для RNN.

2.3. Tesseract OCR

Tesseract – OCR-двигун з відкритим кодом, який є дуже популярним серед розробників OCR.

Tesseract сумісний з багатьма мовами програмування та фреймворками через спеціальні обгортки. Він може бути використаний у парі з аналізом наявного макета для розпізнавання тексту у великому документі. Також його можна використати разом із зовнішнім детектором тексту для розпізнавання тексту із зображення, написавши мінімальну кількість коду.

Tesseract найкраще працює, коли є чітка сегментація тексту на передньому плані відносно його фону. Такі типи налаштування може бути надзвичайно складно гарантувати на практиці. Існує безліч причин, через які Tesseract може не отримати якісний вихідний результат, наприклад, у випадку, коли зображення

має певні шуми на фоні. Чим краща якість зображення (розмір, контраст, освітлення), тим кращий буде результат розпізнавання буде продемонстровано. Це вимагає певної попередньої обробки для поліпшення результатів OCR, зображення потрібно масштабувати належним чином, мати якомога більший контраст зображення, а текст повинен бути вирівняний по горизонталі. Tesseract OCR є досить потужним, але має такі обмеження:

- Tesseract OCR не такий точний, як деякі комерційні рішення, доступні на ринку сьогодні;
- Не вдається показати точне розпізнавання символів при роботі із зображеннями, на які впливають артефакти, такі як часткова оклюзія, спотворена перспектива та складний фон;
- Він не здатний розпізнавати рукописний текст;
- Неякісні скани можуть призвести до низькоякісного розпізнавання.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ОТРИМАНІ РЕЗУЛЬТАТИ

3.1. Створення набору зображень

Для реалізації поставленої задачі було створено два набори зображень у відтінках сірого низької якості. Текст на зображення поданий використовуючи 10 різних типів шрифтів:

```
# get list font
font_lst = ['arial', 'arialbd', 'times', 'timesbd', 'timesi','ariblk',
'arialbd', 'arialbi', 'ariali', 'timesbi']
```

Перший набір зображень складається з 4960 зображень розміру 250x50 пікселів, які містять випадково нанесені великі й маленькі букви англійського алфавіту та цифри з розміром шрифту 24px.

Другий набір зображення складається з 4010 зображень розміру 680x50 пікселів, які містять шматки змістовного тексту з книги «The Hunger Games». Розмір шрифту 24 px.

Спершу для тренування та тестування OCR системи було використано перший набір зображень, до яких застосовано функції розмиття та нанесення цифрового шуму, що описано вище в 2 розділі. Потім таким же чином був використаний другий набір зображень.

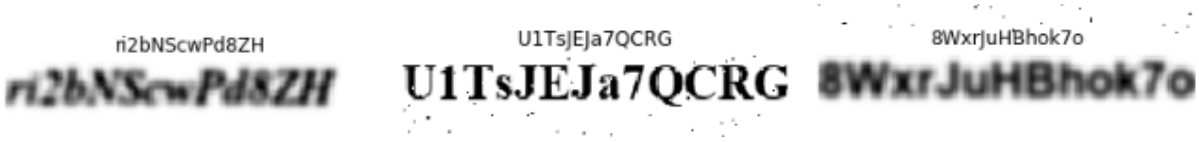

Рис. 3.1. Приклади зображень із першого набору після спотворення

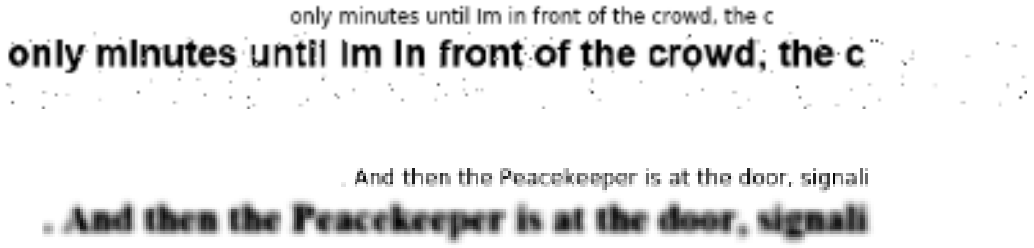

Рис. 3.2. Приклади зображень із другого набору після спотворення

3.2. Розробка моделі для розпізнавання тексту

Модель для розпізнавання тексту реалізована на основі статті «OCR model for reading Captchas»[5]. Цей приклад демонструє просту модель OCR, створену за допомогою функцій API, в якій є поєднання CNN і RNN. А також він ілюструє, як можна створити екземпляр нового рівня та використовувати його як «рівень кінцевої точки» для реалізації втрати CTC.

API функції для побудови моделі зображені на рисунку 3.3 у колонці під назвою «Layer (type)».

```
Model: "ocr_model_v1"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
=================================================================================================
 image (InputLayer)          [(None, 200, 50, 1)       0           []
                             ]

 Conv1 (Conv2D)              (None, 200, 50, 32)       320         ['image[0][0]']

 pool1 (MaxPooling2D)        (None, 100, 25, 32)       0           ['Conv1[0][0]']

 Conv2 (Conv2D)              (None, 100, 25, 64)       18496       ['pool1[0][0]']

 pool2 (MaxPooling2D)        (None, 50, 12, 64)        0           ['Conv2[0][0]']

 reshape (Reshape)           (None, 50, 768)           0           ['pool2[0][0]']

 dense1 (Dense)              (None, 50, 64)            49216       ['reshape[0][0]']

 dropout_2 (Dropout)         (None, 50, 64)            0           ['dense1[0][0]']

 bidirectional_4 (Bidirectional  (None, 50, 256)       197632      ['dropout_2[0][0]']
 )

 bidirectional_5 (Bidirectional  (None, 50, 128)       164352      ['bidirectional_4[0][0]']
 )

 label (InputLayer)          [(None, None)]            0           []

 dense2 (Dense)              (None, 50, 64)            8256        ['bidirectional_5[0][0]']

 ctc_loss (CTCLayer)         (None, 50, 64)            0           ['label[0][0]',
                                                                    'dense2[0][0]']

=================================================================================================
Total params: 438,272
Trainable params: 438,272
Non-trainable params: 0
```

Рис. 3.3. – Модель для першого набору зображень

```
Model: "ocr_model_v1"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================================
 image (InputLayer)             [(None, 680, 50, 1)  0           []
                                ]

 Conv1 (Conv2D)                 (None, 680, 50, 32)  320         ['image[0][0]']

 pool1 (MaxPooling2D)           (None, 340, 25, 32)  0           ['Conv1[0][0]']

 Conv2 (Conv2D)                 (None, 340, 25, 64)  18496       ['pool1[0][0]']

 pool2 (MaxPooling2D)           (None, 170, 12, 64)  0           ['Conv2[0][0]']

 reshape (Reshape)              (None, 170, 768)     0           ['pool2[0][0]']

 dense1 (Dense)                 (None, 170, 64)      49216       ['reshape[0][0]']

 dropout (Dropout)              (None, 170, 64)      0           ['dense1[0][0]']

 bidirectional (Bidirectional)  (None, 170, 256)     197632      ['dropout[0][0]']

 bidirectional_1 (Bidirectional  (None, 170, 128)    164352      ['bidirectional[0][0]']
 )

 label (InputLayer)             [(None, None)]       0           []

 dense2 (Dense)                 (None, 170, 73)      9417        ['bidirectional_1[0][0]']

 ctc_loss (CTCLayer)            (None, 170, 73)      0           ['label[0][0]',
                                                                  'dense2[0][0]']

=================================================================================================
Total params: 439,433
Trainable params: 439,433
Non-trainable params: 0
_________________________________________________________________________________________________
```

Рис. 3.4. – Модель для другого набору зображень

3.3. Тренування та валідація моделі

Набори зображень, що були створені для тренування та валідації розробленої моделі містять зображення низької якості, які поділені у співвідношенні 9:1 відповідно.

Тренування моделі на першому наборі зображень відбувалося протягом 70 епох в середньому за 124 секунди кожна. Статистику тренування моделі наведено у Додатку А. Значення функції втрат під час тренування та валідації наведено на рисунку 3.5.

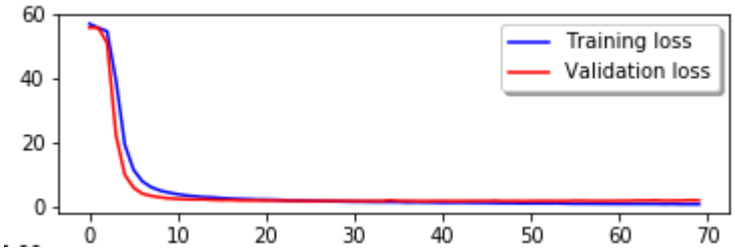

Рис. 3.5. Значення функції втрат

Тренування моделі на другому наборі зображень відбувалося протягом 80 епох в середньому за 246 секунд кожна. Статистику тренування моделі наведено у Додатку В. Значення функції втрат під час тренування та валідації наведено на рисунку 3.6.

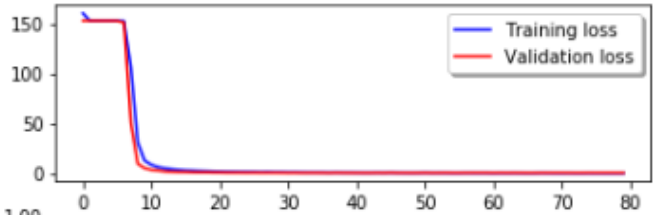

Рис. 3.6. Значення функції втрат

3.4. Результати та порівняння із Tesseract OCR

Результати роботи моделі на першому наборі зображень представлені на рисунках 3.7, 3.8 та 3.9 (результати роботи розпізнавання).

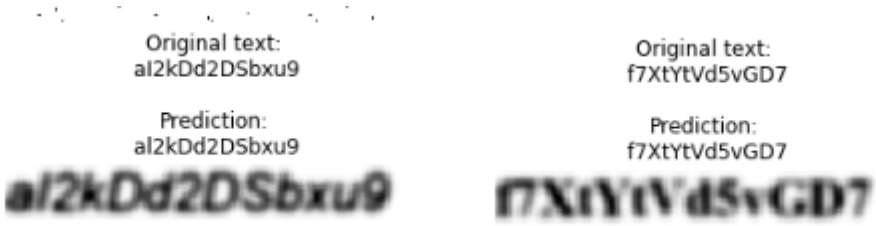

Рис. 3.7. Результати розпізнавання

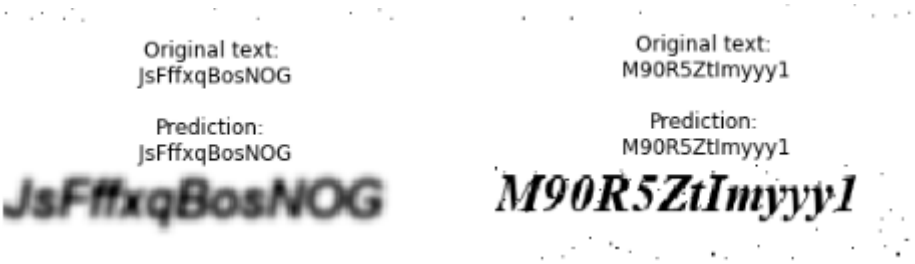

Рис. 3.8. Результати розпізнавання

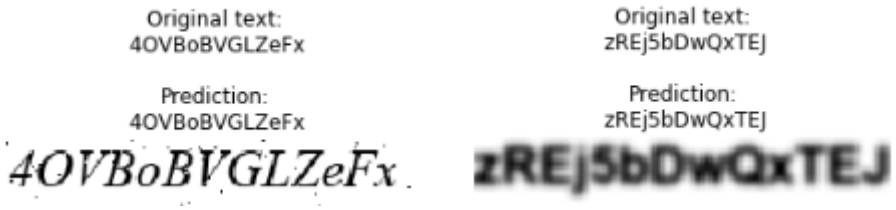

Рис. 3.9. Результати розпізнавання

Результати роботи моделі на другому наборі зображень представлені на рисунках 3.10, 3.11, 3.12 та 3.13.

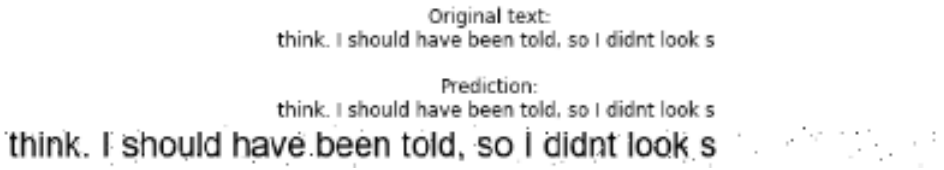

Рис. 3.10. Результати розпізнавання

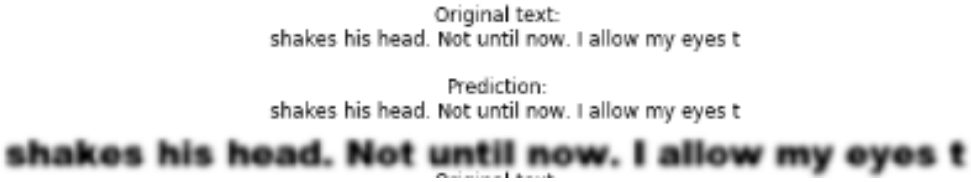

Рис. 3.11. Результати розпізнавання

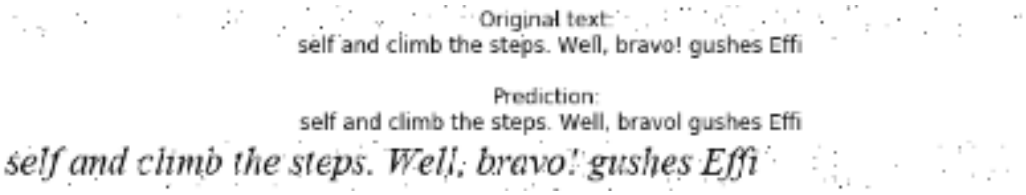

Рис. 3.12. Результати розпізнавання

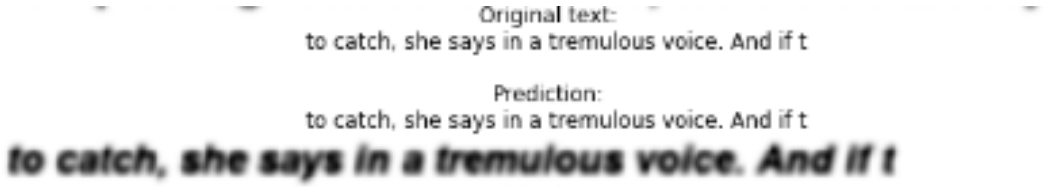

Рис. 3.13. Результати розпізнавання

Отримані моделі можуть розпізнавати приблизно 68.75% зображень на першому валідаційному наборі зображень та 81.25% на другому валідаціному наборі.

Для порівняння було використано Tesseract OCR. Результати розпізнавання тексту зображені на рисунках 3.14, 3.15, 3.16, 3.17 нижче.

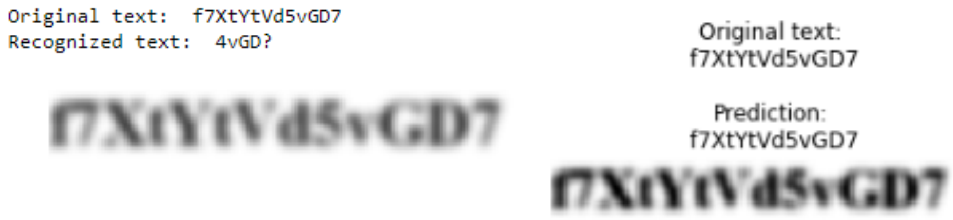

Рис. 3.14. Результати розпізнавання Tesseract OCR і розробленої моделі

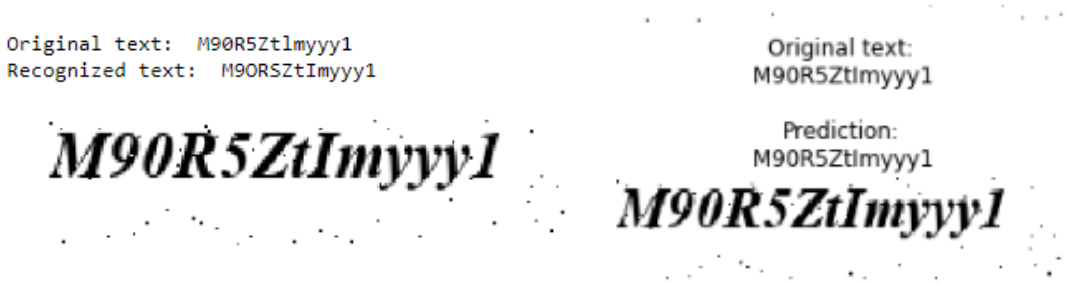

Рис. 3.15. Результати розпізнавання Tesseract OCR і розробленої моделі

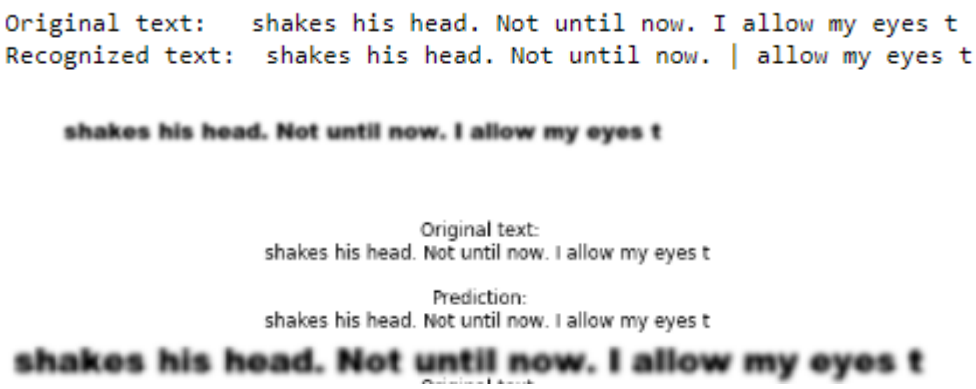

Рис. 3.16. Результати розпізнавання Tesseract OCR і розробленої моделі

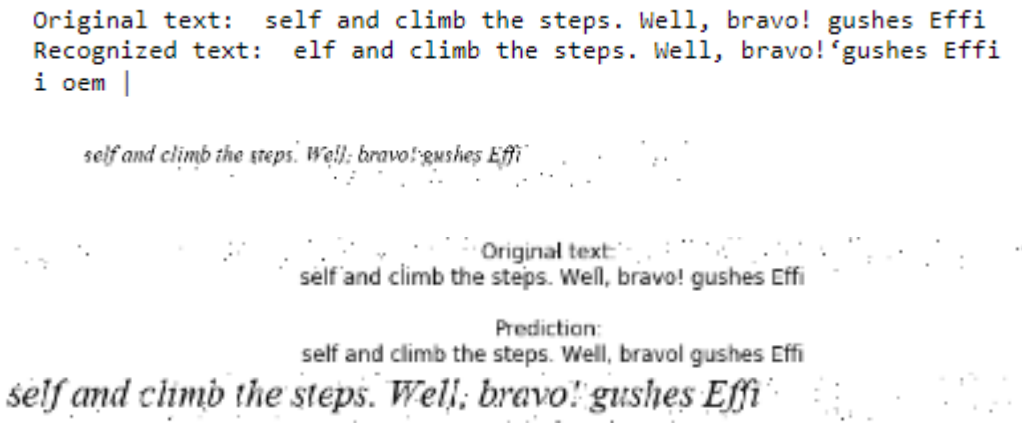

Рис. 3.17. Результати розпізнавання Tesseract OCR і розробленої моделі

Виходячи із результатів на рисунках, то можна зробити висновок, що система Tesseract OCR доволі погано працює із неякісними зображеннями. Базуючись на моїх спостереженнях у розпізнаванні тексту на спотворених зображеннях, то зображення із цифровим шумом розпізнаються краще, ніж розмитті зображення.

ВИСНОВКИ

У даній роботі було реалізовано систему для розпізнавання незмістовного та змістовного тексту використовуючи методи машинного навчання. Дані моделі було натреновано та протестовано на специфічних наборах даних, які містять спотворені зображення у різний спосіб. Частина зображень спотворена використовуючи розмиття, а інша частина – використовуючи цифровий шум. Під час роботи було проаналізовано існуючі системи для розпізнавання друкованого тексту, проблеми розпізнавання друкованого тексту, розглянуто методи розпізнавання тексту.

Дане програмне забезпечення може знайти продовження для розпізнавання змістовного багаторядкового тексту , що в сучасних реаліях є дуже корисним, адже люди надають перевагу збереженню текстової інформацію на своїх смартфонах, де зображення та текст може бути спотворений.

СПИСОК ЛІТЕРАТУРИ

1. OCR. Режим доступу: https://uk.wikipedia.org/wiki/Оптичне_розпізнавання_символів
2. Переяславська, С., Шевченко, В., & Смагіна, О. (2021). АНАЛІЗ ПІДХОДІВ ДО РОЗПІЗНАВАННЯ ТЕКСТОВОЇ ІНФОРМАЦІЇ У ТЕХНОЛОГІЇ OCR. Режим доступу: https://ojs.ukrlogos.in.ua/index.php/interconf/article/view/9924
3. Методи розпізнавання тексту. Режим доступу: https://uk.wikipedia.org/wiki/Методи_розпізнавання_тексту
4. Neural Networks. Режим доступу: https://www.ibm.com/cloud/learn/neural-networks
5. OCR model. Режим доступу: https://keras.io/examples/vision/captcha_ocr/
6. OCR system. Режим доступу: https://www.analyticsvidhya.com/blog/2020/05/build-your-own-ocr-google-tesseract-opencv
7. How to use PIL. Режим доступу: https://pillow.readthedocs.io/en/stable/
8. How to use OpenCv. Режим доступу: https://pypi.org/project/opencv-python/
9. RNN neural network. Режим доступу: https://en.wikipedia.org/wiki/Recurrent_neural_network
10. Згорткові нейронні мережі. Режим доступу: https://evergreens.com.ua/ua/articles/cnn.html
11. Цифровий шум. Режим доступу: https://medium.com/image-vision/noise-in-digital-image-processing-55357c9fab71#:~:text=There%20are%20three%20types%20of,value)%20all%20over%20the%20image.
12. Розмиття зображення. Режим доступу: https://www.geeksforgeeks.org/what-is-image-blurring/
13. Neural Networks: A Comprehensive Foundation Simon Haykin 842 pages

ДОДАТОК А

```
Epoch 1/70
279/279 [==============================] - 143s 440ms/step - loss: 57.0190 -
val_loss: 55.8409
Epoch 2/70
279/279 [==============================] - 110s 392ms/step - loss: 55.8012 -
val_loss: 55.8039
Epoch 3/70
279/279 [==============================] - 110s 393ms/step - loss: 54.7026 -
val_loss: 51.0488
Epoch 4/70
279/279 [==============================] - 111s 398ms/step - loss: 39.4361 -
val_loss: 22.3570
Epoch 5/70
279/279 [==============================] - 109s 392ms/step - loss: 19.3988 -
val_loss: 9.9839
Epoch 6/70
279/279 [==============================] - 100s 360ms/step - loss: 11.4975 -
val_loss: 5.9349
Epoch 7/70
279/279 [==============================] - 113s 406ms/step - loss: 7.9075 -
val_loss: 4.0739
Epoch 8/70
279/279 [==============================] - 114s 407ms/step - loss: 6.1023 -
val_loss: 3.4060
Epoch 9/70
279/279 [==============================] - 101s 363ms/step - loss: 5.0515 -
val_loss: 2.9508
Epoch 10/70
279/279 [==============================] - 510s 2s/step - loss: 4.4201 - val
_loss: 2.6248
Epoch 11/70
279/279 [==============================] - 167s 599ms/step - loss: 3.9098 -
val_loss: 2.4687
Epoch 12/70
279/279 [==============================] - 179s 641ms/step - loss: 3.5210 -
val_loss: 2.3384
Epoch 13/70
279/279 [==============================] - 189s 677ms/step - loss: 3.2733 -
val_loss: 2.2874
Epoch 14/70
279/279 [==============================] - 188s 672ms/step - loss: 3.0347 -
val_loss: 2.3202
Epoch 15/70
279/279 [==============================] - 176s 631ms/step - loss: 2.9334 -
val_loss: 2.1533
Epoch 16/70
279/279 [==============================] - 190s 680ms/step - loss: 2.7171 -
val_loss: 2.0822
Epoch 17/70
279/279 [==============================] - 181s 649ms/step - loss: 2.5410 -
val_loss: 2.1430
Epoch 18/70
```

```
279/279 [==============================] - 201s 721ms/step - loss: 2.4444 -
val_loss: 1.9854
Epoch 19/70
279/279 [==============================] - 193s 693ms/step - loss: 2.3613 -
val_loss: 1.9782
Epoch 20/70
279/279 [==============================] - 182s 651ms/step - loss: 2.2532 -
val_loss: 2.0026
Epoch 21/70
279/279 [==============================] - 185s 664ms/step - loss: 2.2377 -
val_loss: 1.9548
Epoch 22/70
279/279 [==============================] - 200s 718ms/step - loss: 2.1679 -
val_loss: 1.9671
Epoch 23/70
279/279 [==============================] - 164s 588ms/step - loss: 2.0682 -
val_loss: 1.9037
Epoch 24/70
279/279 [==============================] - 195s 698ms/step - loss: 1.9674 -
val_loss: 1.8896
Epoch 25/70
279/279 [==============================] - 191s 685ms/step - loss: 1.9398 -
val_loss: 1.8180
Epoch 26/70
279/279 [==============================] - 172s 615ms/step - loss: 1.8500 -
val_loss: 1.8398
Epoch 27/70
279/279 [==============================] - 285s 1s/step - loss: 1.7867 - val
_loss: 1.8391
Epoch 28/70
279/279 [==============================] - 283s 1s/step - loss: 1.7503 - val
_loss: 1.8453
Epoch 29/70
279/279 [==============================] - 187s 670ms/step - loss: 1.7061 -
val_loss: 1.8512
Epoch 30/70
279/279 [==============================] - 183s 658ms/step - loss: 1.6374 -
val_loss: 1.8148
Epoch 31/70
279/279 [==============================] - 203s 728ms/step - loss: 1.5735 -
val_loss: 1.8454
Epoch 32/70
279/279 [==============================] - 159s 571ms/step - loss: 1.5902 -
val_loss: 1.7899
Epoch 33/70
279/279 [==============================] - 160s 574ms/step - loss: 1.5211 -
val_loss: 1.8294
Epoch 34/70
279/279 [==============================] - 158s 567ms/step - loss: 1.5096 -
val_loss: 1.7819
Epoch 35/70
279/279 [==============================] - 158s 568ms/step - loss: 1.5005 -
val_loss: 2.0545
Epoch 36/70
279/279 [==============================] - 159s 569ms/step - loss: 1.4993 -
val_loss: 1.8371
```

```
Epoch 37/70
279/279 [==============================] - 167s 598ms/step - loss: 1.3704 -
val_loss: 1.8538
Epoch 38/70
279/279 [==============================] - 160s 572ms/step - loss: 1.4181 -
val_loss: 1.7915
Epoch 39/70
279/279 [==============================] - 159s 570ms/step - loss: 1.3773 -
val_loss: 1.7865
Epoch 40/70
279/279 [==============================] - 702s 3s/step - loss: 1.3358 - val
_loss: 1.8046
Epoch 41/70
279/279 [==============================] - 121s 433ms/step - loss: 1.2651 -
val_loss: 1.8375
Epoch 42/70
279/279 [==============================] - 118s 424ms/step - loss: 1.2872 -
val_loss: 1.7955
Epoch 43/70
279/279 [==============================] - 128s 458ms/step - loss: 1.3137 -
val_loss: 1.8056
Epoch 44/70
279/279 [==============================] - 128s 457ms/step - loss: 1.2768 -
val_loss: 1.8171
Epoch 45/70
279/279 [==============================] - 122s 436ms/step - loss: 1.2427 -
val_loss: 1.8474
Epoch 46/70
279/279 [==============================] - 131s 469ms/step - loss: 1.2218 -
val_loss: 1.8195
Epoch 47/70
279/279 [==============================] - 127s 456ms/step - loss: 1.1771 -
val_loss: 1.9146
Epoch 48/70
279/279 [==============================] - 123s 439ms/step - loss: 1.1455 -
val_loss: 1.7736
Epoch 49/70
279/279 [==============================] - 123s 441ms/step - loss: 1.1469 -
val_loss: 1.8194
Epoch 50/70
279/279 [==============================] - 127s 454ms/step - loss: 1.0848 -
val_loss: 1.7582
Epoch 51/70
279/279 [==============================] - 121s 432ms/step - loss: 1.0794 -
val_loss: 1.8057
Epoch 52/70
279/279 [==============================] - 118s 424ms/step - loss: 1.0968 -
val_loss: 1.8322
Epoch 53/70
279/279 [==============================] - 124s 443ms/step - loss: 1.0686 -
val_loss: 1.8357
Epoch 54/70
279/279 [==============================] - 127s 456ms/step - loss: 1.1306 -
val_loss: 1.7951
Epoch 55/70
```

```
279/279 [==============================] - 127s 456ms/step - loss: 1.0211 -
val_loss: 1.8193
Epoch 56/70
279/279 [==============================] - 131s 471ms/step - loss: 0.9638 -
val_loss: 1.9054
Epoch 57/70
279/279 [==============================] - 125s 448ms/step - loss: 0.9814 -
val_loss: 1.8926
Epoch 58/70
279/279 [==============================] - 119s 428ms/step - loss: 0.9596 -
val_loss: 1.8250
Epoch 59/70
279/279 [==============================] - 103s 370ms/step - loss: 0.9327 -
val_loss: 1.8293
Epoch 60/70
279/279 [==============================] - 76s 271ms/step - loss: 0.9289 - v
al_loss: 1.8527
Epoch 61/70
279/279 [==============================] - 91s 325ms/step - loss: 0.9113 - v
al_loss: 1.8648
Epoch 62/70
279/279 [==============================] - 76s 272ms/step - loss: 0.9066 - v
al_loss: 1.8701
Epoch 63/70
279/279 [==============================] - 79s 284ms/step - loss: 0.9205 - v
al_loss: 1.9786
Epoch 64/70
279/279 [==============================] - 78s 278ms/step - loss: 0.9145 - v
al_loss: 1.9478
Epoch 65/70
279/279 [==============================] - 85s 306ms/step - loss: 0.9244 - v
al_loss: 2.0486
Epoch 66/70
279/279 [==============================] - 74s 263ms/step - loss: 0.8426 - v
al_loss: 1.9040
Epoch 67/70
279/279 [==============================] - 71s 255ms/step - loss: 0.8949 - v
al_loss: 1.9219
Epoch 68/70
279/279 [==============================] - 71s 254ms/step - loss: 0.8157 - v
al_loss: 1.9641
Epoch 69/70
279/279 [==============================] - 71s 254ms/step - loss: 0.8215 - v
al_loss: 2.0550
Epoch 70/70
279/279 [==============================] - 71s 254ms/step - loss: 0.8126 - v
al_loss: 2.0048
```

ДОДАТОК Б

```
Epoch 1/80
226/226 [==============================] - 186s 791ms/step - loss: 160.4578 - va
l_loss: 153.0514
Epoch 2/80
226/226 [==============================] - 177s 783ms/step - loss: 152.9778 - va
l_loss: 153.0110
Epoch 3/80
226/226 [==============================] - 177s 785ms/step - loss: 152.9486 - va
l_loss: 152.9880
Epoch 4/80
226/226 [==============================] - 178s 786ms/step - loss: 152.9383 - va
l_loss: 152.9648
Epoch 5/80
226/226 [==============================] - 177s 785ms/step - loss: 152.9304 - va
l_loss: 152.9536
Epoch 6/80
226/226 [==============================] - 176s 780ms/step - loss: 152.9221 - va
l_loss: 152.9267
Epoch 7/80
226/226 [==============================] - 178s 787ms/step - loss: 152.6397 - va
l_loss: 151.0885
Epoch 8/80
226/226 [==============================] - 177s 782ms/step - loss: 106.2822 - va
l_loss: 50.2029
Epoch 9/80
226/226 [==============================] - 178s 786ms/step - loss: 31.4411 - val
_loss: 10.1665
Epoch 10/80
226/226 [==============================] - 177s 784ms/step - loss: 13.3647 - val
_loss: 5.6654
Epoch 11/80
226/226 [==============================] - 177s 784ms/step - loss: 9.0617 - val_
loss: 3.9777
Epoch 12/80
226/226 [==============================] - 178s 787ms/step - loss: 6.6817 - val_
loss: 3.1522
Epoch 13/80
226/226 [==============================] - 177s 784ms/step - loss: 5.4657 - val_
loss: 2.3997
Epoch 14/80
226/226 [==============================] - 177s 785ms/step - loss: 4.5325 - val_
loss: 2.0232
Epoch 15/80
```

```
226/226 [==============================] - 179s 791ms/step - loss: 3.9402 - val_
loss: 1.7855
Epoch 16/80
226/226 [==============================] - 181s 800ms/step - loss: 3.3887 - val_
loss: 1.7480
Epoch 17/80
226/226 [==============================] - 178s 787ms/step - loss: 3.1340 - val_
loss: 1.5065
Epoch 18/80
226/226 [==============================] - 179s 792ms/step - loss: 2.8107 - val_
loss: 1.4136
Epoch 19/80
226/226 [==============================] - 178s 788ms/step - loss: 2.5949 - val_
loss: 1.4147
Epoch 20/80
226/226 [==============================] - 179s 792ms/step - loss: 2.3251 - val_
loss: 1.2613
Epoch 21/80
226/226 [==============================] - 178s 790ms/step - loss: 2.1343 - val_
loss: 1.2158
Epoch 22/80
226/226 [==============================] - 178s 787ms/step - loss: 1.9476 - val_
loss: 1.0905
Epoch 23/80
226/226 [==============================] - 178s 790ms/step - loss: 1.8636 - val_
loss: 1.0744
Epoch 24/80
226/226 [==============================] - 178s 788ms/step - loss: 1.7550 - val_
loss: 1.0578
Epoch 25/80
226/226 [==============================] - 179s 792ms/step - loss: 1.7180 - val_
loss: 0.9941
Epoch 26/80
226/226 [==============================] - 178s 788ms/step - loss: 1.6515 - val_
loss: 1.0501
Epoch 27/80
226/226 [==============================] - 178s 789ms/step - loss: 1.4867 - val_
loss: 0.9680
Epoch 28/80
226/226 [==============================] - 179s 791ms/step - loss: 1.6004 - val_
loss: 0.9898
Epoch 29/80
226/226 [==============================] - 179s 790ms/step - loss: 1.4022 - val_
loss: 0.9723
Epoch 30/80
```

```
226/226 [==============================] - 179s 792ms/step - loss: 1.4130 - val_
loss: 0.9343
Epoch 31/80
226/226 [==============================] - 179s 793ms/step - loss: 1.3019 - val_
loss: 0.8708
Epoch 32/80
226/226 [==============================] - 178s 790ms/step - loss: 1.2922 - val_
loss: 0.9196
Epoch 33/80
226/226 [==============================] - 178s 790ms/step - loss: 1.2020 - val_
loss: 0.8569
Epoch 34/80
226/226 [==============================] - 252s 1s/step - loss: 1.1400 - val_los
s: 0.8873
Epoch 35/80
226/226 [==============================] - 304s 1s/step - loss: 1.1304 - val_los
s: 0.8975
Epoch 36/80
226/226 [==============================] - 309s 1s/step - loss: 1.1106 - val_los
s: 0.8704
Epoch 37/80
226/226 [==============================] - 352s 2s/step - loss: 1.0448 - val_los
s: 0.8148
Epoch 38/80
226/226 [==============================] - 297s 1s/step - loss: 1.0582 - val_los
s: 0.9170
Epoch 39/80
226/226 [==============================] - 305s 1s/step - loss: 0.9978 - val_los
s: 0.8753
Epoch 40/80
226/226 [==============================] - 307s 1s/step - loss: 1.0061 - val_los
s: 0.8501
Epoch 41/80
226/226 [==============================] - 306s 1s/step - loss: 0.9687 - val_los
s: 0.8271
Epoch 42/80
226/226 [==============================] - 304s 1s/step - loss: 0.9197 - val_los
s: 0.8261
Epoch 43/80
226/226 [==============================] - 1375s 6s/step - loss: 0.9271 - val_lo
ss: 0.8152
Epoch 44/80
226/226 [==============================] - 286s 1s/step - loss: 0.9404 - val_los
s: 0.8809
Epoch 45/80
```

```
226/226 [==============================] - 306s 1s/step - loss: 0.8983 - val_los
s: 0.8366
Epoch 46/80
226/226 [==============================] - 307s 1s/step - loss: 0.9106 - val_los
s: 0.9502
Epoch 47/80
226/226 [==============================] - 303s 1s/step - loss: 0.8760 - val_los
s: 0.8264
Epoch 48/80
226/226 [==============================] - 304s 1s/step - loss: 0.8274 - val_los
s: 0.8329
Epoch 49/80
226/226 [==============================] - 305s 1s/step - loss: 0.8134 - val_los
s: 0.8086
Epoch 50/80
226/226 [==============================] - 307s 1s/step - loss: 0.8062 - val_los
s: 0.8416
Epoch 51/80
226/226 [==============================] - 305s 1s/step - loss: 0.8755 - val_los
s: 0.8820
Epoch 52/80
226/226 [==============================] - 305s 1s/step - loss: 0.8169 - val_los
s: 0.8225
Epoch 53/80
226/226 [==============================] - 305s 1s/step - loss: 0.8217 - val_los
s: 0.8313
Epoch 54/80
226/226 [==============================] - 306s 1s/step - loss: 0.7655 - val_los
s: 0.8949
Epoch 55/80
226/226 [==============================] - 308s 1s/step - loss: 0.7281 - val_los
s: 0.8314
Epoch 56/80
226/226 [==============================] - 306s 1s/step - loss: 0.7327 - val_los
s: 0.8311
Epoch 57/80
226/226 [==============================] - 305s 1s/step - loss: 0.7798 - val_los
s: 0.8542
Epoch 58/80
226/226 [==============================] - 305s 1s/step - loss: 0.7313 - val_los
s: 0.8991
Epoch 59/80
226/226 [==============================] - 306s 1s/step - loss: 0.7081 - val_los
s: 0.9011
Epoch 60/80
```

```
226/226 [==============================] - 307s 1s/step - loss: 0.7299 - val_los
s: 0.8739
Epoch 61/80
226/226 [==============================] - 305s 1s/step - loss: 0.6460 - val_los
s: 0.8360
Epoch 62/80
226/226 [==============================] - 305s 1s/step - loss: 0.6928 - val_los
s: 0.8812
Epoch 63/80
226/226 [==============================] - 305s 1s/step - loss: 0.6824 - val_los
s: 0.8368
Epoch 64/80
226/226 [==============================] - 305s 1s/step - loss: 0.7078 - val_los
s: 0.8657
Epoch 65/80
226/226 [==============================] - 305s 1s/step - loss: 0.6673 - val_los
s: 0.8429
Epoch 66/80
226/226 [==============================] - 307s 1s/step - loss: 0.6469 - val_los
s: 0.8013
Epoch 67/80
226/226 [==============================] - 309s 1s/step - loss: 0.6529 - val_los
s: 0.8069
Epoch 68/80
226/226 [==============================] - 209s 924ms/step - loss: 0.7353 - val_
loss: 0.8395
Epoch 69/80
226/226 [==============================] - 187s 828ms/step - loss: 0.6733 - val_
loss: 0.8567
Epoch 70/80
226/226 [==============================] - 181s 799ms/step - loss: 0.6409 - val_
loss: 0.8788
Epoch 71/80
226/226 [==============================] - 180s 797ms/step - loss: 0.6019 - val_
loss: 0.8883
Epoch 72/80
226/226 [==============================] - 180s 797ms/step - loss: 0.6499 - val_
loss: 0.8361
Epoch 73/80
226/226 [==============================] - 180s 795ms/step - loss: 0.5950 - val_
loss: 0.8352
Epoch 74/80
226/226 [==============================] - 183s 808ms/step - loss: 0.5917 - val_
loss: 0.8336
Epoch 75/80
```

```
226/226 [==============================] - 179s 794ms/step - loss: 0.6116 - val_
loss: 0.8739
Epoch 76/80
226/226 [==============================] - 180s 796ms/step - loss: 0.5814 - val_
loss: 0.8723
Epoch 77/80
226/226 [==============================] - 180s 795ms/step - loss: 0.5657 - val_
loss: 0.8843
Epoch 78/80
226/226 [==============================] - 183s 808ms/step - loss: 0.5489 - val_
loss: 0.9005
Epoch 79/80
226/226 [==============================] - 179s 792ms/step - loss: 0.5664 - val_
loss: 0.8684
Epoch 80/80
226/226 [==============================] - 181s 801ms/step - loss: 0.6106 - val_
loss: 0.8523
```